Dérivation d'algorithmes distribués d'arbitrage

Jean-Pierre Verjus, René Thoraval

INRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 414

DÉRIVATION D'ALGORITHMES DISTRIBUÉS D'ARBITRAGE

Jean-Pierre VERJUS
René THORAVAL

Juin 1985

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 - RENNES CÉDEX
FRANCE
Tél. : (99) 36.20.00
Télex : UNIRISA 95 0473 F

Publication interne n° 255

Avril 1985

30 pages

DERIVATION D'ALGORITHMES DISTRIBUES D'ARBITRAGE

Jean-Pierre VERJUS
IRISA
Université de Rennes I
Campus de Beaulieu
35042 - RENNES Cédex

René THORAVAL
Département de Mathématiques
et Informatique
Université de Nantes
44072 - NANTES Cédex

RESUME :

A partir d'une expression abstraite très simple définissant un protocole d'arbitrage entre processus, on dérive des algorithmes par transformations élémentaires et on retrouve entre autres de nombreux algorithmes distribués d'exclusion mutuelle ou plus généralement d'allocation de ressource.

ABSTRACT :

We show how process arbitration protocol algorithms can be derived from very simple abstract expression by means of elementary transformations. In particular, a number of mutual exclusion distributed algorithms and more generally, resource allocation algorithms can be rediscovered by this method.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L.A. 227)
UNIVERSITÉ DE RENNES 1 I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

TABLE DES MATIERES

MOTS-CLEFS :

Algorithmes distribués, dérivation d'algorithmes, compteurs de synchronisation, arbitrage, exclusion mutuelle, allocation de ressources, problème des 5 philosophes.

DERIVATION D'ALGORITHMES DISTRIBUES D'ARBITRAGE

Depuis une dizaine d'années ont été proposés d'assez nombreux algorithmes distribués sans que leur(s) auteur(s) prennent forcément le soin de donner quelques pistes sur les principes de construction qu'ils ont utilisés. Carvalho et Roucairol [CAR 83] ont proposé un procédé de construction d'algorithmes distribués à partir d'assertion. Bochman [BOC 79], André, Herman et Verjus [voir en particulier AHV 80, HER 83, VER 83] montrent que des expressions abstraites de synchronisation définies en terme des compteurs de Robert et Verjus [ROV 77] peuvent avoir des caractéristiques régulières qui en favorisent la répartition. Dans de nombreux cas, ces compteurs sont monotones croissants et peuvent se mettre en oeuvre Modulo N, ce qui est une propriété nécessaire quand on travaille sur des machines réelles.

L'article qui suit commence par un exercice pédagogique destiné à illustrer la méthode qu'on peut appeler "De l'expression abstraite à la mise en oeuvre distribuée par dérivation" d'algorithmes écrits avec des compteurs monotones. A partir de l'exemple très simple choisi, on retrouve, par transformations élémentaires des algorithmes d'arbitrage (ou d'exclusion mutuelle) dont ceux proposés par Dijkstra [DIJ 74], Kessels [KES 82], Raynal [RAY 85].

On montre alors que ces formes d'algorithmes appartiennent à une même variété dont la forme proposée par Dijkstra [DIJ 74] a un caractère générique. Cette forme générique, qui plus est, permet de composer des algorithmes d'allocation de ressources plus généraux que le simple problème d'arbitrage.

1 - ARBITRAGE BINAIRE

1.1. EXPRESSION ABSTRAITE DU PROBLEME

On considère deux processus P_0 et P_1 qui entrent en conflit pour l'accès à une section critique.

On veut définir un protocole d'arbitrage tel que l'exécution parallèle de P_0 et P_1 engendre l'une des deux suites infinies :

$$SC_0 \; ; \; SC_1 \; ; \; SC_0 \; ; \; SC_1 \; ; \; \ldots$$

ou

$$SC_1 \; ; \; SC_0 \; ; \; SC_1 \; ; \; SC_0 \; ; \; \ldots$$

où SC_i ($i \in \{0,1\}$) représente le passage de P_i en section critique.

Un tel protocole assure l'accès exclusif d'un processus à la section critique. Il est équitable puisqu'il assure une alternance de SC_0 et SC_1 et sans blocage puisque la suite engendrée est infinie.

1.2. CONTRAINTES DE MISE EN OEUVRE

On veut mettre en oeuvre ce protocole dans un système réparti composé de deux sites, avec un processus par site. Pour cela, on se fixe les contraintes suivantes :

C_1 : il n'y a pas de variable partagée par les deux processus.

C_2 : chaque processus peut lire et modifier ses variables locales mais ne peut que lire celles de l'autre processus.

1.3. USAGE DE COMPTEURS MONOTONES

Dans un système centralisé, l'intérêt d'utiliser des compteurs pour exprimer des contraintes de synchronisation a été largement montré [ROV 77, AHV 80]. Par ailleurs, dans un système distribué, l'utilisation de compteurs monotones croissants présente deux avantages essentiels (voir [REK 79], [AHV 80], [COR 81], [HER 83], [VER 83]).

Le premier avantage est que sur tout site on peut connaître une borne inférieure de la valeur d'un compteur évoluant sur un autre site, quel que soit le délai de transmission (fini bien sûr). L'usage d'une telle borne inférieure est souvent suffisante pour évaluer une condition de synchronisation. Ainsi, considérons 2 entiers croissants a et l tels que $0 \leq a - l \leq n$ (par exemple a matérialise le nombre d'accès à une ressource, l le nombre de libérations et n le nombre total de ressources). Soient deux sites distants S_a et S_l sur lesquels on connaît respectivement la valeur exacte de a et l. Un contrôleur, localisé sur le site S_a doit, avant d'incrémenter la variable a (c'est-à-dire, en suivant notre analogie, laisser utiliser une ressource), vérifier que :

$$a - l < n \qquad (1)$$

Or, sur ce site, on ne connaît pas la valeur exacte de l, mais un minorant $\bar{l}$. Cependant, le contrôleur peut s'assurer que

$$a - \bar{l} < n$$

ce qui lui garantira que la condition théorique (1) est bien vérifiée.

Le second avantage des compteurs croissants est que, lorsque leur évolution différentielle est bornée (par exemple ici $0 \leq a - l \leq n$) et que leur croissance s'effectue par pas de 1, on peut les mettre en oeuvre modulo K (K = différence maximum + 1), ici n+1, et en déduire la nouvelle contrainte.

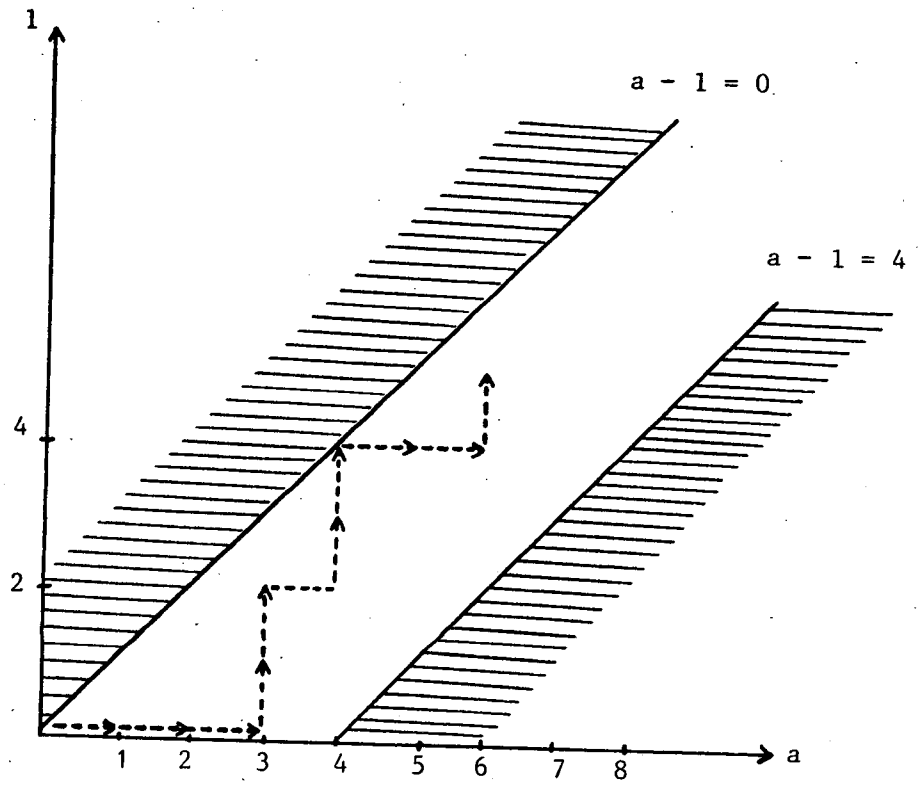
Si A et L désignent les valeurs modulo (n+1) de a et l, les contraintes à faire respecter, sont dans notre exemple :

Condition pour augmenter A : $A - L \neq n$ modulo n+1
(ce qui est équivalent à $A - L \neq -1$).

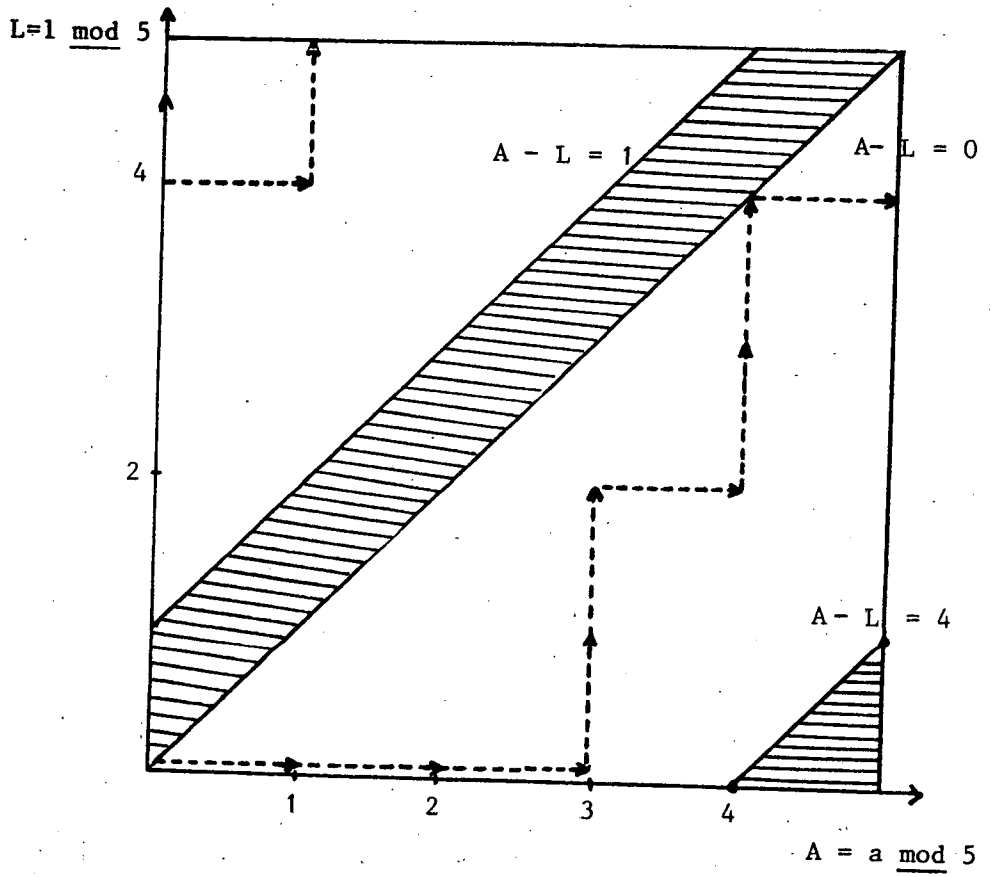
Condition pour augmenter L : $A - L \neq 0$
(Cette condition est vérifiée par construction si toute libération suit une attribution).

Les deux schémas qui suivent illustrent cette propriété.

Contrainte $0 \leq a - 1 \leq 4$



Exemple d'évolution possible



Carré d'évolution $(N \bmod 5)^2$.

Evolution des compteurs modulo correspondants à l'exemple de gauche.

Les zones interdites sont hachurées.

En conclusion, on peut donc considérer que l'usage de compteurs croissants permet de s'affranchir (s'abstraire) des contraintes dues aux délais de transmission et de celles dues à la mise en oeuvre avec des variables informatiques.

1.4. DEUX SOLUTIONS AVEC UN ORDRE CHOISI A-PRIORI

Dans le paragraphe précédent, nous avons montré que l'expression d'un problème en termes de compteurs monotones est une abstraction tout à fait commode dont il est facile de donner une mise en oeuvre distribuée. Reprenons le problème posé en 1.1. pour le traduire en terme de compteurs. Dans ce cas, il est naturel de considérer deux compteurs x0 et x1, initialisés à 0, servant à comptabiliser le nombre de passages de P_0 et P_1 en section critique et on exprime la condition d'accès de P_i à sa section critique en fonction de ces compteurs.

L'accès exclusif à la section critique impose de fixer des conditions d'accès contradictoires pour P_0 et P_1, ce qui garantit par la même occasion l'absence de blocage. Assurer l'alternance entre P_0 et P_1 implique d'interdire à un processus de prendre plus d'un tour d'avance sur son voisin. On aboutit donc naturellement aux conditions suivantes :

$$(1)\quad \begin{cases} \text{Condition d'accès } (P_0) \equiv (x_0 - x_1 = 0) \\ \text{Condition d'accès } (P_1) \equiv (x_0 - x_1 = 1) \end{cases}$$

ou

$$(2)\quad \begin{cases} \text{Condition d'accès } (P_0) \equiv (x_1 - x_0 = 1) \\ \text{Condition d'accès } (P_1) \equiv (x_1 - x_0 = 0). \end{cases}$$

Il est bien entendu que P_i, à l'issue de sa section critique, donne le témoin à $P_{i\oplus 1}$ en rendant vraie la condition d'accès de $P_{i\oplus 1}$.

On constate que les compteurs x_0 et x_1 sont tels que $0 \leq x_0 - x_1 \leq 1$ (ou $0 \leq x_1 - x_0 \leq 1$), ce qui autorise à les mettre en oeuvre modulo 2. Soient X_0 et X_1 la représentation modulo 2 de x_0 et x_1.

Le choix des conditions (1) permet d'aboutir à l'algorithme (I) qui est d'ailleurs celui de Dijkstra [DIJ 74], et où P_0 est favorisé au départ.

(I)

```
P0 | X0 := 0                          P1 | X1 := 0
   | répéter                             | répéter
   |   attendre X0 = X1                  |   attendre X0 = X1 ⊕ 1
   |   SC0                               |   SC1
   |   X0 := X0 ⊕ 1                      |   X1 := X1 ⊕ 1
   | fin répéter                          | fin répéter
```

où ⊕ représente l'addition modulo 2.

Le choix des conditions (2) permet d'obtenir, quant à lui, l'algorithme (II), où P_1 est le premier servi :

(II)

```
P0 | X0 := 0                          P1 | X1 := 0
   | répéter                             | répéter
   |   attendre X1 = X0 ⊕ 1              |   attendre X0 = X1
   |   SC0                               |   SC1
   |   X0 := X0 ⊕ 1                      |   X1 := X1 ⊕ 1
   | fin répéter                          | fin répéter
```

1.5. COMMENTAIRE SUR LA FORME DONNEE AUX PROCESSUS

Il est à noter que les deux processus P_0 et P_1 sont supposés, au départ, en situation de communiquer, c'est-à-dire que P_i peut accéder à la variable de $P_{i\oplus 1}$. Par ailleurs, nous n'avons pas exprimé ici le fait que le processus puisse avoir à exécuter des actions spécifiques en dehors de la section critique. En réalité, il faut considérer que P_0 et P_1 sont les

processus chargés d'établir le protocole d'arbitrage pour le compte de processus en compétition. Ainsi, SC_i doit être considéré comme la représentation d'une sollicitation émise par un processus, sur le site n° i, pour entrer en section critique. Si cette sollicitation n'existe pas, tout se passe comme si SC_i était l'instruction vide, c'est-à-dire que P_i passe immédiatement le témoin à $P_{i\oplus 1}$.

Ainsi, le protocole d'arbitrage met en oeuvre deux processus actifs en permanence, c'est-à-dire en situation de communiquer entre eux comme avec le (ou les) processus du site à qui chacun fournit l'autorisation d'entrer en section critique.

1.6. PROBLEME DE LA PRIORITE INITIALE

Dans les deux expressions données en 1.4., on favorise a-priori un processus. On réalise en fait un ordonnancement statique des processus, puis cet ordre est répété infiniment.

On peut s'affranchir de ce choix initial en considérant que les deux variables X_0 et X_1 peuvent avoir une valeur quelconque dans {0,1}. On a alors un ordre aléatoire, mais on notera que X_0 et X_1 ne sont plus exactement la représentation modulo 2 du nombre de passages en section critique.

Un autre choix serait d'établir un ordre Premier arrivé - Premier servi (PAPS). Dans ce but, on peut s'inspirer du protocole habituel de politesse qui régit le franchissement d'une porte lorsque deux personnes s'y présentent simultanément.

Ce protocole les conduit à ne pas chercher à forcer le passage, mais au contraire à se le céder. Chacun, en quelque sorte, exécute un algorithme qui se caractérise par un arrêt marqué, l'esquisse d'un geste et l'attente d'un signal. Afin que le protocole aboutisse, on peut décider des règles suivantes : le "premier" arrivé esquisse un geste pour faire passer le "second" ; celui-ci s'efface alors et donne le passage au premier qui s'exécute. Pour la personne n° i (i=0 ou 1), cela se traduit ainsi

```
PRIORITE := i ⊕ 1                    / ⊕ addition modulo 2 /
attendre PRIORITE = i
entrer
```

Cette règle peut se traduire ainsi dans les formulations (I) ou (II) : devant la boucle de chacun des P_i, on remplace l'initialisation de X_i à 0 par une instruction qui rend vraie la condition d'accès de $P_{i\oplus 1}$ à sa section critique. On dérive ainsi de (I) la solution (I') où la priorité initiale est de type PAPS :

(I')

```
P0 | X0 := X1 ⊕ 1                 P1 | X1 := X0
   | répéter                          | répéter
   |   attendre X0 = X1               |   attendre X0 = X1 ⊕ 1
   |   SC0                            |   SC1
   |   X0 := X0 ⊕ 1                   |   X1 := X1 ⊕ 1
   | fin répéter                       | fin répéter
```

On constate alors qu'on peut fusionner les instructions :

$X_0 := X_1 \oplus 1$ avec $X_0 := X_0 \oplus 1$

$X_1 := X_0$ avec $X_1 := X_1 \oplus 1$

et on obtient finalement la solution "PAPS" (III) :

(III)

```
P0 | répéter                      P1 | répéter
   |   X0 := X1 ⊕ 1                 |   X1 := X0
   |   attendre X0 = X1              |   attendre X0 = X1 ⊕ 1
   |   SC0                           |   SC1
   | fin répéter                     | fin répéter
```

De façon analogue, on peut dériver de (II) la solution PAPS (IV) :

(IV)

```
P0 | répéter                          P1 | répéter
   |   X0 = X1                           |   X1 := X0 ⊕ 1
   |   attendre X1 = X0 ⊕ 1              |   attendre X1 ≠ X0
   |   SC0                               |   SC1
   | fin répéter                         | fin répéter
```

Notons que la formulation (3) ci-dessous constitue donc une abstraction des algorithmes (III) et (IV) :

(3)

```
Pi | répéter
   |    PRIORITE := i ⊕ 1
   |    attendre PRIORITE = i
   |    SCi
   | fin répéter
```

et on peut énoncer :

(1) L'instruction initiale PRIORITE := i ⊕ 1 établit un rendez-vous entre P_0 et P_1, sur la première exécution de attendre.

(2) Ce rendez-vous réalisé, on a, ∀i :
condition pour autoriser SC_i : PRIORITE = i
action à la terminaison de $SC_{i\oplus 1}$: PRIORITE := i

ce qui constitue bien un énoncé abstrait du comportement souhaité pour P_0 // P_1 dans 1.1.

On peut dériver (III) et (IV) de la formulation (3) en considérant que les compteurs de passage en section critique doivent mettre en oeuvre la variable théorique PRIORITE. Dans ce but, on effectue les changements de variables suivants :

Pour dériver III : PRIORITE = $x_0 - x_1$

ou encore

$= (x_0 \oplus x_1)$

Pour dériver IV : PRIORITE = $(x_0 - x_1) \oplus 1$

ou encore

$= (x_0 \oplus x_1 \oplus 1)$

Ces changements de variable traduisent bien le fait qu'un processus a la priorité dès qu'il a un tour de retard sur (resp. le même nombre de tour que) son voisin.

Remarque :

On retrouve dans la forme (3) ci-dessus une version élaguée de l'algorithme centralisé d'exclusion mutuelle donné par Peterson [PET 81], et dans la forme (IV), une version élaguée de l'algorithme distribué donné par Kessels [KES 82].

A titre d'illustration, donnons l'algorithme de Kessels :

```
P_i : répéter
        Q[i] := vrai
        R[i] := R[i⊕1] ⊕ i
        attendre non Q[i⊕1] ou R[i] ≠ R[i⊕1] ⊕ i
        SC_i
        Q[i] := faux
      fin répéter
```

Q[i] = vrai signifie que P_i est présent dans la compétition (dans le cas contraire, il est déclaré perdant par forfait). Si on fait abstraction de cette possibilité de forfait, on est dans les conditions de nos hypothèses, et on retrouve l'algorithme (IV),rendu, bien artificiellement, symétrique.

1.7. PROTOCOLE D"OUVERTURE"

Les expressions (I) à (IV) supposent que les variables d'un processus soient accessibles à l'autre processus dès qu'il démarre. Ainsi, il faut considérer un protocole de mise en route (ouverture de session par exemple) : un processus particulier envoie un signal à tous les processus. Notons que dans la formulation (I) ou (II), ce protocole d'ouverture doit en outre conduire à l'initialisation des variables à 0 si on a choisi un ordonnancement statique (cf. 1.6.).

1.8. CONCLUSION

Nous avons donc dérivé plusieurs algorithmes d'arbitrage binaire et montré que deux aspects doivent être examinés

- Initialisation des variables
- Choix de l'ordre à établir entre les deux processus

la répétition par alternance de ce choix initial étant ensuite un problème assez élémentaire.

2 - ARBITRAGE ENTRE N PROCESSUS PAR CIRCULATION D'UN PRIVILÈGE SUR UN ANNEAU

La généralisation du protocole d'arbitrage à n sites va se poser de la même manière qu'au paragraphe précédent (voir 1.8). On doit examiner principalement l'initialisation des variables et la manière d'établir un ordre entre les processus. Il existe de nombreuses stratégies selon qu'un processus sera autorisé à communiquer avec un, plusieurs ou tous les autres processus. L'anneau est la structure de communication la plus simple. Les n processus sont numérotés de 0 à n-1 et un processus ne communique qu'avec son "voisin" sur l'anneau virtuel constitué par cette numérotation a-priori. Nous développons dans ce paragraphe des algorithmes distribués d'arbitrage sur un anneau. Dans le paragraphe 4, nous évoquerons deux autres structures de communication : le maillage total et l'arbre binaire. Auparavant, dans le paragraphe 3, nous évoquerons d'autres algorithmes distribués sur un anneau.

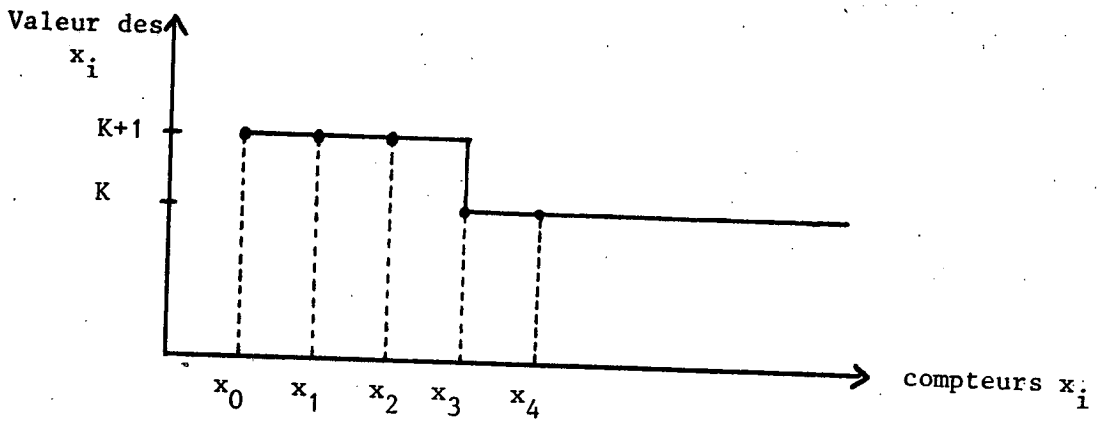
2.1. ABSTRACTION

Considérons n processus P_i, $i \in [0, n-1]$, rangés en anneau. Nous n'allons considérer ici qu'une seule méthode de passage de témoin d'un processus à l'autre : le processus P_i cède le privilège à $P_{i \boxplus 1}$ ($\boxplus$ = addition modulo n). Le problème, exprimé avec la variable PRIORITE est simple :

(4)
```
Pi : | répéter
     |   attendre PRIORITE = i
     |     SCi
     |   PRIORITE := i ⊞ 1
     | fin répéter
```

PRIORITE ayant au départ une valeur donnée, 0 par exemple.

Exprimer ce protocole avec des compteurs croissants x_i qui comptabilisent les passages en section critique est simple. En effet, l'évolution du système peut se représenter par le "dénivellé" suivant :



dans lequel P_3 a le privilège (c'est-à-dire peut entrer en section critique).

Si, au départ, toutes les variables x_i sont nulles, le système évolue de la manière suivante :

(5)
$$\text{PRIORITE} = i \iff x_{i-1} = x_i + 1 \ , \ i=1, \ldots n-1$$
$$= 0 \iff x_0 = x_{n-1}$$
$$\text{PRIORITE} := i \boxplus 1 \iff x_i := x_i + 1$$

Comme au paragraphe 3, on note que $0 \leq x_{i \ominus 1} - x_i \leq 1$ ($\ominus$ soustraction modulo n) et on en déduit que x_i peut être mis en oeuvre modulo 2.

2.2. ALGORITHME DE BASE

On obtient alors l'énoncé (V), qui est celui de Dijkstra [DIJ 74] et qui généralise l'énoncé (I). X_i est la représentation de x_i modulo 2.

(V)

$X_i \in \{0,1\}$, $i \in \{0, \ldots, n-1\}$

P_0 :
```
X0 := 0
répéter
  attendre X0 = Xn-1
  SC0
  X0 := X0 ⊕ 1
fin répéter
```

P_i $i \neq 0$ =
```
Xi := 0
répéter
  attendre Xi = Xi-1 ⊕ 1
  SCi
  Xi := Xi ⊕ 1
fin répéter
```

De la même manière, on peut aboutir à un algorithme symétrique du précédent, qui généralise l'énoncé (II): P_{n-1} joue le rôle de P_0 et le privilège parcourt l'anneau de P_i vers $P_{i\ominus 1}$.

2.3. PROBLEME DE LA PRIORITE INITIALE

Dans le § 1.6., nous avons montré pour deux processus que le mode d'initialisation des variables X_0 et X_1 induisait trois types de politique d'attribution de la priorité initiale :

- Ordonnancement statique, par initialisation des variables,
- Ordonnancement aléatoire, par non initialisation,
- Ordonnancement PAPS, par initialisation croisée.

De la même manière, pour un nombre quelconque de processus, l'algorithme (V) réalise un ordonnancement statique par initialisation des variables à 0. D'une façon plus générale :

- Toute initialisation des X_i ne créant pas de dénivellé donne au départ le privilège à P_0.

- Toute initialisation des X_i créant un seul dénivellé donne au départ la priorité au processus où apparaît le dénivellé.

Par contre, une non-initialisation n'a pas le même effet : la cohérence du système n'est pas garantie (on trouvera dans [MTV 77] l'établissement des conditions qui garantissent que le privilège circulera dans de bonnes conditions au bout d'un temps fini ; mais cette question est hors du champ de notre étude).

Quant à l'ordonnancement PAPS, il est clair qu'il ne peut être généralisé sans qu'un processus entre en compétition, donc en communication, directement ou non avec tous les autres processus : on sort donc du cadre de l'anneau et nous y reviendrons dans un paragraphe ultérieur (4).

2.4. VARIETES D'ALGORITHMES DE CIRCULATION D'UN PRIVILEGE SUR UN ANNEAU

Dans l'algorithme (V), le privilège est matérialisé par un dénivellé, ou front (sauf quand P_0 le possède, à charge pour lui de le réengendrer). En quelque sorte, tous les processus sont à égalité au départ ($X_i = 0$, $\forall i$), et celui qui a le privilège est le premier à constater qu'il a du retard sur son voisin (sauf s'il s'agit de P_0 qui constate, quant à lui, que P_{n-1} l'a rattrapé).

Exemple : La figure 1 décrit l'évolution de la configuration de l'anneau avec n = 5. Sur chaque dessin, en abscisse on trouve le numéro du processus (de 0 à 4), en ordonnée la valeur de la variable X_i (X_i = 0 ou 1).
* marque le processus qui a le privilège.

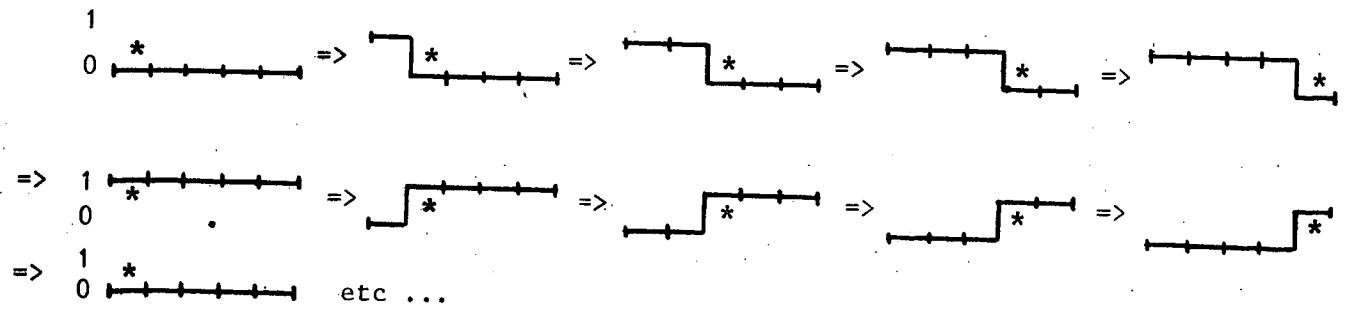


Fig. 1 - Circulation du privilège (*). Algorithme (V), n=5

Sur ce thème, on peut effectuer différentes variations. L'idée de base est de placer au départ, tous les processus dans une situation différente : tous les couples de voisins présentent alors un décalage K analogue ($K \neq 0$) sauf un qui présente un décalage K' singulier ($K' \neq K$, K' éventuellement nul). Il suffit ensuite de rendre cette situation cyclique et infinie.

Prenons par exemple K=1 et K'=0. On obtient alors l'algorithme (VI), qu'on représentera symboliquement comme la forme (5) de l'algorithme (V).

(VI)

X_i = i modulo p, au départ $\forall i \in \{0, \ldots, n-1\}$

n > 2, p étant n'importe quel diviseur de n-1. ($\neq 1$).

PRIORITE = i $\iff X_{i \boxminus 1} = X_i$

PRIORITE := i + 1 $\iff X_i := X_i + 1$ modulo p

La condition d'attribution du privilège ne peut être valable au départ que pour X_0. Donc au départ $X_{n-1} = n-1$ modulo p = $X_0 = 0$, ce qui implique que p soit un diviseur de n-1. On vérifie aisément que la système évolue de manière cohérente. Lorsque n est impair, on peut donc prendre p=2 et on obtient une structure intéressante, en "créneau". (Voir figure 2).

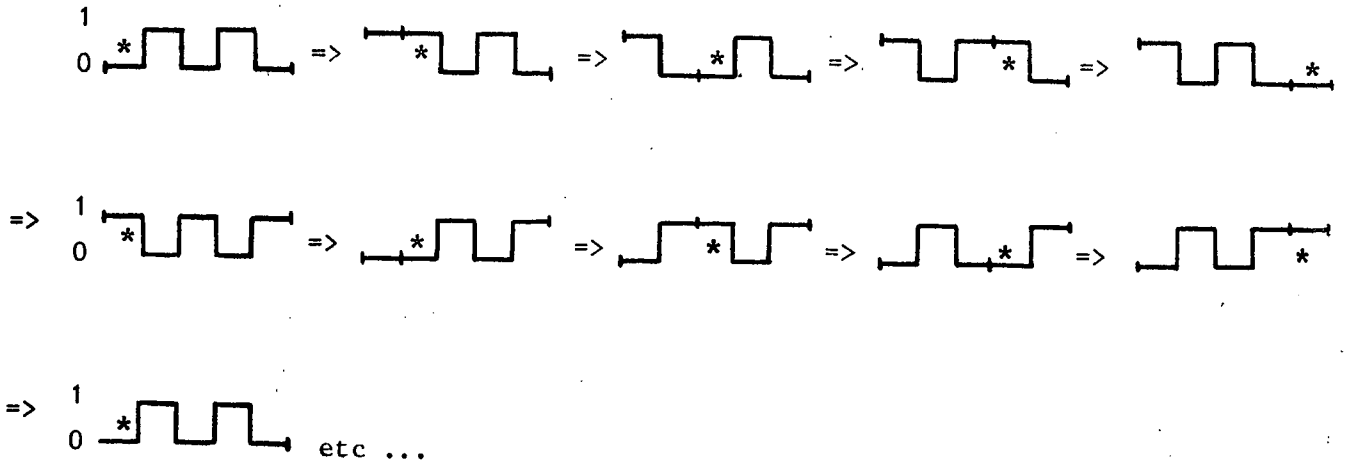


Fig. 2 - Circulation du privilège (*). Algorithme (VI), n=5 (addition modulo 2).

Si on choisit K=1 et K'=1-n (conditions initiales), on obtient l'algorithme de Raynal [RAY 85]. Dans [RAY 85], $p=n^2$: en réalité il est nécessaire et suffisant que p ne divise pas n. (Si p divise n, alors K = K' modulo p et tous les couples présentent la même configuration). Cet algorithme est le suivant :

(VII)

X_i = i modulo p, au départ, $\forall i \in \{0, \ldots n-1\}$

n > 1, p n'est pas diviseur de n.

PRIORITE = i $\Longleftrightarrow$ $X_{i-1} - X_i = n-1$ modulo p

PRIORITE := i $\boxplus$ 1 $\Longleftrightarrow$ $X_i := X_i + n$ modulo p

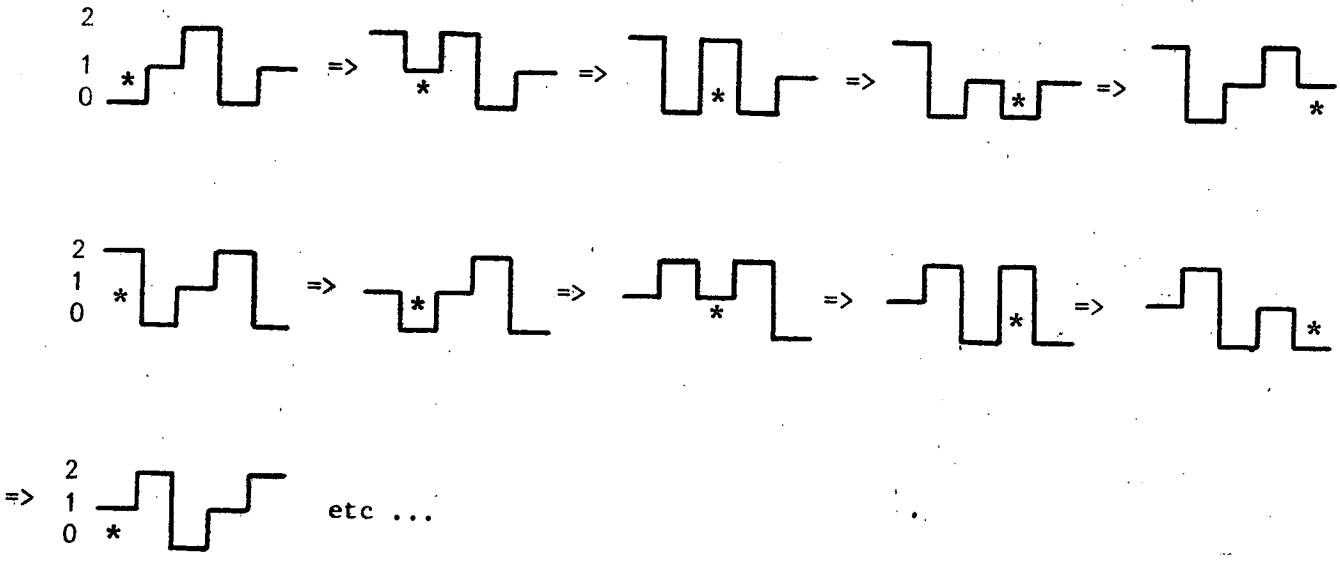


Fig. 3 - Circulation du privilège (*), algorithme (VII), n = 5 (Addition modulo 3).

Mais comme nous allons le voir, tous ces algorithmes ont une base commune. En effet, grâce au changement de variable

(6) $\quad Y_i = n * X_i + i \qquad \forall i \in \{0, \ldots, n-1\}$

on dérive de (V) l'algorithme (VII).

	V	VII
Initialisation	$X_i := 0 \quad \forall i$	$Y_i := i \quad \forall i$
Priorité = 0	$X_{n-1} - X_0 = 0$	$Y_{n-1} - Y_0 = n-1$
Priorité = i $i \neq 0$	$X_{i-1} - X_i = 1$	$Y_{i-1} - Y_i = n + (i-1) - i$ $= n - 1$
Priorité := i ⊞ 1	$X_i := X_i + 1$	$Y_i := Y_i + n$

De plus, quand n > 2, si p divise (n-1), alors p ne divise pas n. Donc, dès que n > 2, on peut gérer les variables Y_i de l'algorithme (VII) modulo p, où p est un diviseur de n-1. On a donc n-1 = 0 modulo p et on obtient l'algorithme (VI). (Voir Figure 4).

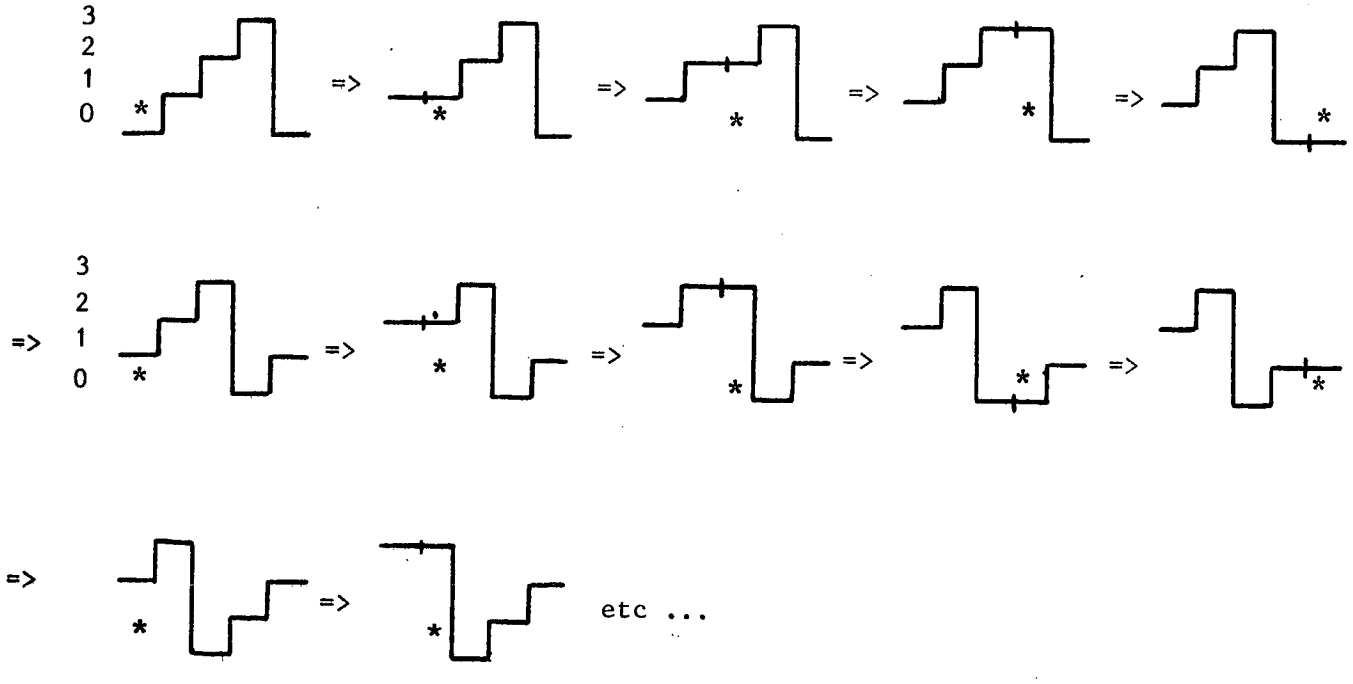


Fig. 4 - Circulation du privilège (*) - Algorithme (VI) ou (VII). n = 5 (addition modulo 4).

En fait, on peut encore généraliser (VII) en modulant certains paramètres tels que K, le numéro du premier processus privilégié ... Nous pensons pouvoir avancer que toute forme ainsi obtenue est une variété, dérivée de la forme régulière (V) par un changement de variable de type (6).

3. PRODUCTION D'ALGORITHMES DISTRIBUÉS ÉQUITABLES D'ALLOCATION DE RESSOURCES

On peut envisager de faire circuler 2 (resp. p) privilèges parmi n processus P_i selon le modèle développé au paragraphe 2. Cette méthode est une manière élégante de résoudre l'allocation équitable de 2 ressources (resp. p) à n processus. Rappelons que dans ce cas, comme nous l'avons déjà noté en 1.5., la possession du privilège par un processus doit être comprise commme la possibilité d'attribuer une ressource sur le site correspondant. Le processus P_i a pour seul rôle de distribuer, récupérer et faire circuler les droits d'utilisation des ressources : ces "droits" circulent donc à leur propre vitesse, de manière équitable ; si, sur le site, aucune ressource n'est réclamée par un demandeur, le droit est cédé au site voisin. Notons qu'on ne peut attribuer qu'une ressource à la fois à un demandeur, mais on peut envisager qu'il existe plusieurs demandeurs sur chaque site.

3.1. PRODUCTION PAR ADDITION DE COMPTEURS

Pour faire circuler 2 (resp. p) privilèges, on peut considérer 2 (resp. p) jeux de compteurs indépendants. Mais on observe qu'on peut faire l'addition des 2 (resp. p) jeux de compteurs. Considérons 2 ressources à allouer à n processus et 2 jeux de compteurs évoluant selon (V). Ainsi

. Soit $\{X_i\}$ et $\{Y_i\}$ les deux jeux de compteurs

. Soit $Z_i = X_i + Y_i \quad \forall i$

on a alors : $0 \leq Z_i - Z_{i \ominus 1} \leq 2$

et on en déduit que Z_i peut évoluer modulo 3, selon l'algorithme :

(VIII)

$Z_i = 0$, au départ, $\forall i \in \{0, \ldots, n-1\}$

$\oplus$: addition modulo 3, $n > 1$

P_0 a 2 droits $\Longleftrightarrow Z_{n-1} - Z_0 = 0$

P_0 a 1 droit $\Longleftrightarrow Z_{n-1} - Z_0 = 2$

P_i a 2 droits $\Longleftrightarrow Z_{i-1} - Z_i = 2$, $i > 0$

P_i a 1 droit $\Longleftrightarrow Z_{i-1} - Z_i = 1$, $i > 0$

P_i abandonne 1 droit $\Longleftrightarrow Z_i := Z_i \oplus 1$, $\forall i$

La figure 5 décrit l'évolution du système pour n = 5.

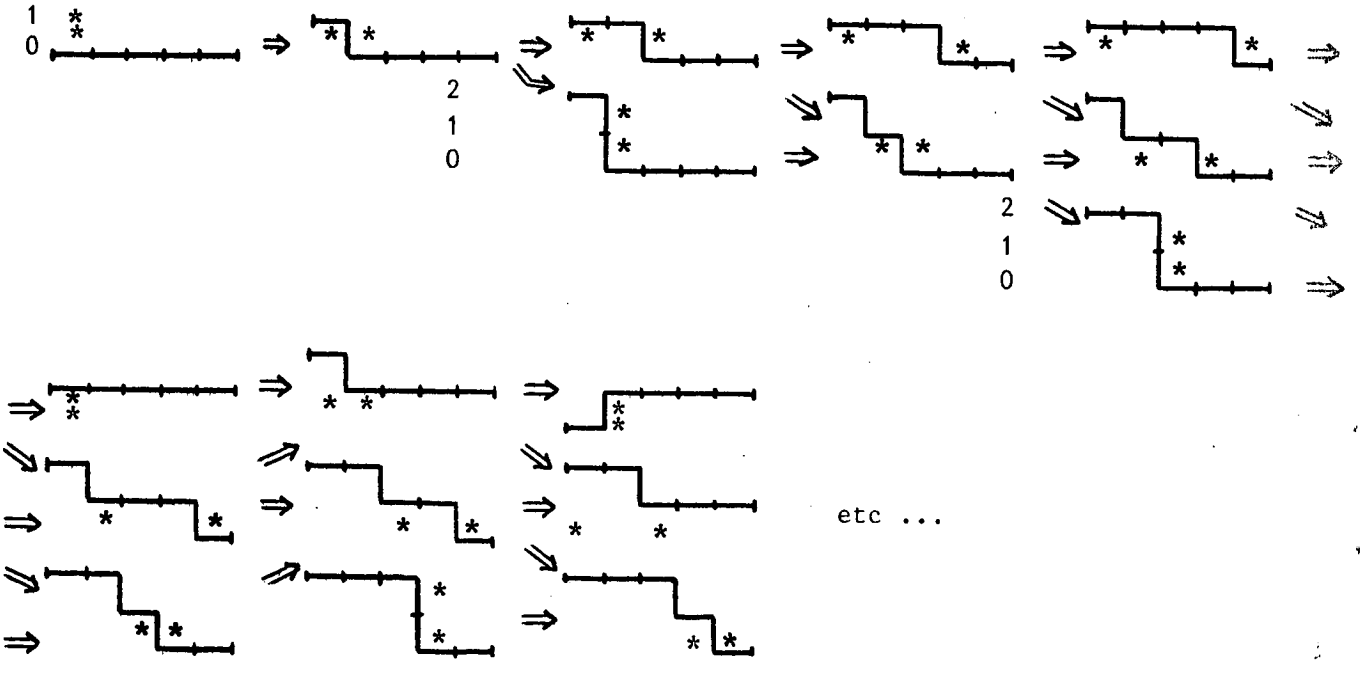


Fig. 5 - Evolution d'un système possédant deux privilèges. n = 5 - Addition modulo 3. (Alg. VIII).

A partir de cette solution, on peut très facilement dériver d'autres solutions à des problèmes d'allocation voisins. Par exemple, si on souhaite ne pas attribuer deux droits simultanés à un même site, on prendra également l'addition de deux jeux de compteurs, en observant :

a) que $0 \leq Z_i - Z_{i \boxminus 1} \leq 1$ => l'addition peut évoluer modulo 2.

b) que P_i, avant de céder un droit qu'il possède , doit vérifier que son voisin de droite ne possède pas déjà un tel droit.

Nous obtenons ainsi un premier algorithme (IX) où chaque site doit interroger ses deux voisins pour prendre une décision. Cet algorithme (IX) est illustré sur la figure 6 (avec n = 5).

(IX)

$i \in \{0, \ldots, n-1\}$, $n > 1$, $\oplus$ addition modulo 2

au départ : $Z_0 = 1$, $Z_i = 0$ pour $i > 0$

. P_i a un droit <=> $Z_i \neq Z_{i \boxminus 1}$

. Condition pour que P_i cède son droit <=> $Z_i = Z_{i \boxplus 1}$

. P_i cède son droit <=> $Z_i := Z_i \oplus 1$

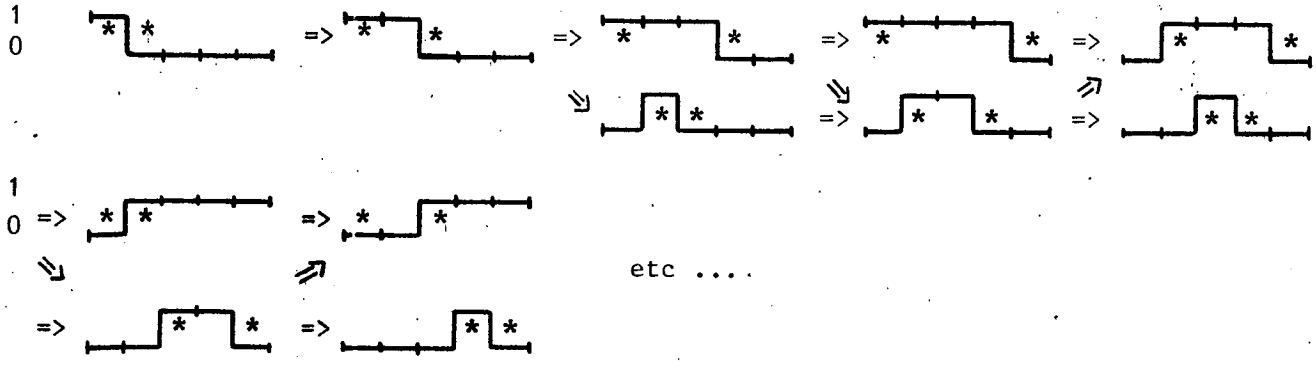


Fig. 6 - Evolution d'un système possédant deux privilèges, un au plus par site. n = 5. Addition modulo 2. Alg. (IX).

A partir de la solution (IX) on peut aisément dériver la solution au problème bien connu des philosophes : en effet, dans ce cas, on considère 5 philosophes dont 2 au plus peuvent disposer d'une ressource si et seulement si leurs deux voisins n'en possèdent pas. Partant de l'algorithme (IX), on constate qu'un processus privilégié $P_{i \boxminus 1}$ peut céder quand il veut son droit : par contre, celui-ci ne sera utilisable par son voisin de droite (soit P_i) que si deux conditions sont satisfaites :

- bien évidemment, le voisin de droite P_i constate que $P_{i \boxminus 1}$ ne l'a plus : $Z_i \neq Z_{i \boxminus 1}$.
- P_i vérifie a son tour que son propre voisin de droite ne l'a pas.

Ce qui donne l'algorithme (X), illustré sur la figure 7 :

(X)

$i \in \{0, 1, 2, 3, 4\}$, $\oplus$ addition modulo 2

au départ : $Z_0 = Z_1 = 1$, $Z_i = 0 \quad i > 1$

P_i a un droit <=> $Z_i \neq Z_{i \boxminus 1}$ et $Z_i = Z_{i \boxplus 1}$

P_i cède son droit <=> $Z_i := Z_i \oplus 1$

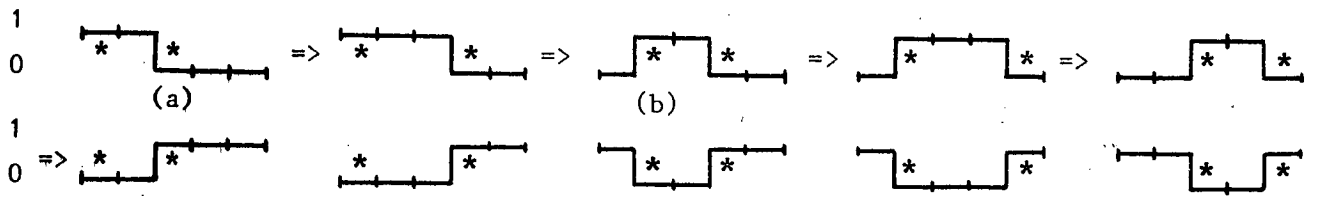
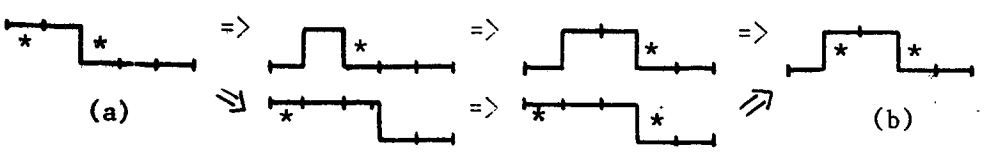
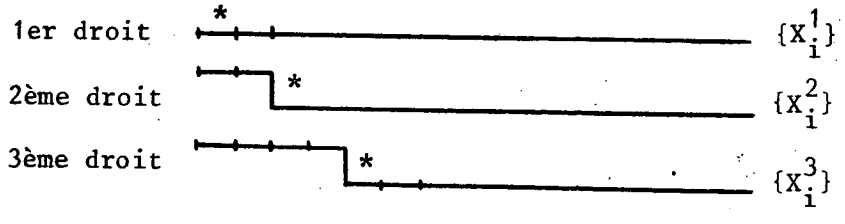


Fig. 7 - Evolution d'un système possédant deux privilèges selon le protocole "5 philosophes". n = 5. Addition modulo 2 - (Alg. (X)). Noter que l'évolution n'est pas exhaustive : ainsi entre l'état (a) et (b) on est susceptible d'avoir



On peut bien sûr généraliser cet algorithme à l'allocation équitable de p ressources à 2 p+1 philosophes (ou même bien sûr à 2 p + q, q < p). L'idée est de considérer p jeux de compteurs $\{X_i^j\}$, $j \in \{1, \ldots, p\}$, $i \in \{0, \ldots, 2p + q - 1\}$ évoluant selon l'algorithme (V) et tels qu'initialement, le privilège soit attribué respectivement aux philosophes 0, 2, 4, ..., p-1.

Exemple :



Par addition, on constate qu'on peut engendrer un jeu de compteurs $\{Z_i\}$, évoluant modulo 2 selon les conditions évoquées en (X).

3.2. - PRODUCTION DIRECTE

On peut obtenir un protocole de type "philosophe" directement, à partir d'un seul jeu de compteurs. Prenons pour simplifier n=5. On associe à chaque philosophe P_i ($i \in \{0, \ldots, 4\}$) un compteur x_i initialisé à 0 et comptabilisant le nombre d'accès de P_i en section critique (droit d'utiliser la ressource). Au départ, tous les x_i sont nuls, et on considère que cette configuration donne un droit à P_0 et à P_2. On en déduit alors que P_i (i pair) a la droit d'utiliser la ressource (on dit en général : de manger) si et seulement si il a eu le même nombre d'occasions que ses voisins (impairs). Lorsque P_0 et P_2 (l'un après l'autre éventuellement) cèdent leur droit, x_0 et x_2 augmentent de 1 et P_1 et P_3 obtiennent le droit de manger, P_1 parce qu'il a un tour de retard sur ses deux voisins, P_3 parce qu'il est en retard avec P_2 et à égalité avec P_4. Quand P_3 abandonne son droit, P_4 constate qu'il a un tour de retard sur chacun de ses voisins et qu'il peut donc manger. A l'issue de son repas éventuel tous les compteurs valent 1. Le protocole peut être réitéré. On a donc

Condition d'attribution du droit à P_0 : $x_0 = x_4 = x_1$

P_2 : $x_2 = x_1 = x_3$

../..

$$P_1 : x_1 = x_0 - 1 = x_2 - 1$$

$$P_4 : x_4 = x_3 - 1 = x_0 - 1$$

$$P_3 : x_3 = x_2 - 1 = x_4$$

Ces compteurs $\{x_i\}$ peuvent être mis en oeuvre modulo 2 (soit $\{X_i\}$) et on obtient un système évoluant comme (X) à un changement de variable près ($X_i \oplus Z_i = C_i$, C_i constante). Voir figure 8.

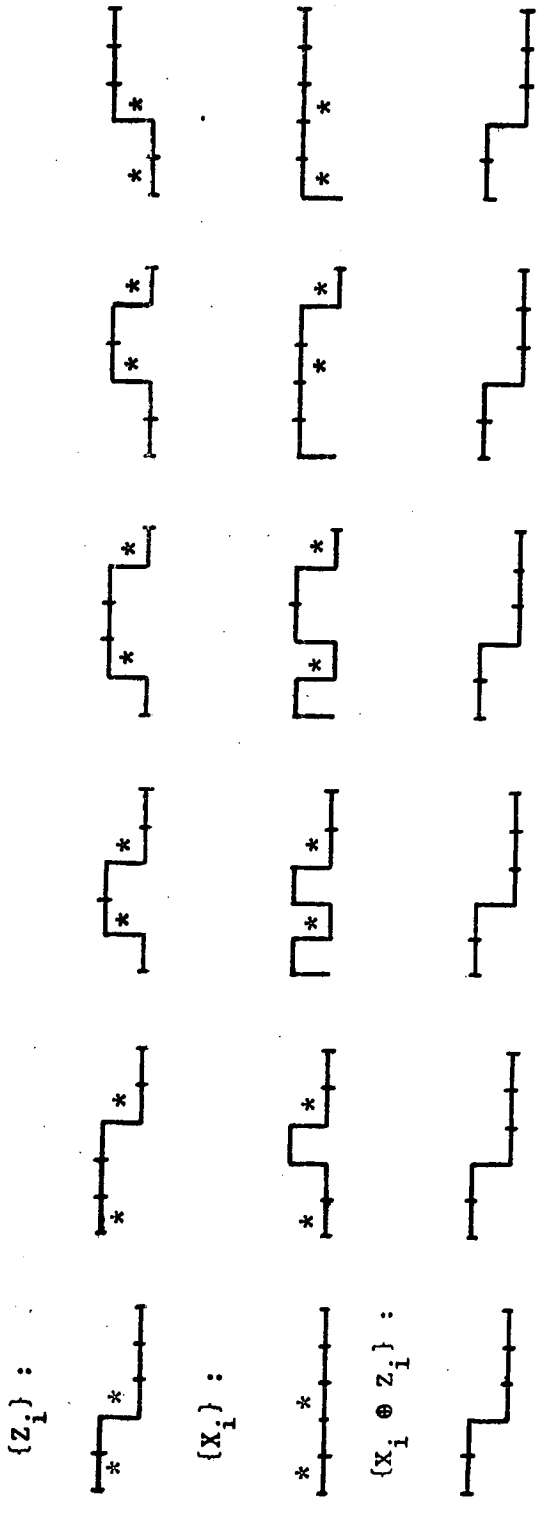


Fig. 8 - Les philosophes : protocole selon (X) ($\{Z_i\}$) et protocole obtenu directement ($\{X_i\}$).

4 - CONCLUSIONS

A partir d'expressions en termes de compteurs croissants, nous avons construit des algorithmes répartis équitables d'allocation de ressources. Dans un premier temps, nous avons retrouvé des algorithmes d'arbitrage par circulation de privilège, et nous en avons dérivé de nouveaux et nous avons montré qu'ils appartenaient tous à une même variété. Dans un second temps, nous avons montré que le mécanisme de circulation de privilège sur un anneau peut à son tour, associé à la définition d'opérations sur les compteurs, être un outil très élégant pour la production d'algorithmes équitables d'allocation de ressources.

STRUCTURE ALGEBRIQUE

Nous pouvons ainsi considérer un jeu de compteurs $\{X_i\}$ évoluant sous les contraintes définies, par exemple, par l'algorithme (V). A partir de là, nous avons donné des exemples de deux expressions

$$(1) \qquad \{Y_i\} = n\,\{X_i\} + a_i$$

$$(2) \qquad \{Z_i\} = \sum_j \{X_i^j\}$$

définissant de nouveaux jeux de compteurs et donnant les bases d'une structure algébrique.

COMPTEURS, PROTOCOLES DE SYNCHRONISATION, ALGORITHMES DISTRIBUES

En ce qui concerne l'expression et la production d'algorithmes distribués, (ou de protocoles), nous avons illustré une fois de plus que les compteurs étaient des outils simples et puissants. Ce sont en effet des outils commodes pour exprimer (d'une manière très condensée) l'histoire passée d'un processus. Leur capacité éventuelle d'évoluer modulo K a une signification précise et un intérêt pratique évident. L'intérêt pratique est celui de la mise en oeuvre informatique. Quant à la signification d'une telle évolution

pour le système décrit, c'est tout simplement le fait qu'à un certain moment, on peut oublier l'histoire passée (on "remet les compteurs à zéro") et réitérer un certain comportement. En quelque sorte, c'est une manifestation du fait que le système décrit est conçu comme la répétition infinie d'un comportement fini et qu'il sera donc sans blocage et sans famine pour peu que le germe (le comportement infiniment répété) ait des propriétés ad'hoc.

INTERET PRATIQUE - ROBUSTESSE

Faire circuler un ou des privilèges sur un anneau pour résoudre des problèmes d'arbitrage et plus généralement d'allocation de ressources relève d'une méthode bien connue qui ne nécessite pas forcément la mise en oeuvre de compteurs évoluant sous contraintes. La circulation d'un message caractéristique est parfois bien suffisante. Bien que ce ne soit pas l'objet de cet article, rappelons (voir [COR 81]) que l'une des justifications des systèmes distribués est leur capacité à résister aux pannes des sites constituants. Faire circuler un message caractéristique présente une faible résistance aux pannes. Par contre, nous avons montré [MTV 77], [COR 81] que l'algorithme (V) présentait une très bonne robustesse. Raynal [RAY 85] évoque également les propriétés de robustesse des algorithmes de type (VII). D'une manière générale, le caractère régulier des contraintes entre les sites permet de déduire les mécanismes de décision ad'hoc en cas de reconfiguration de l'anneau des sites.

AUTRES STRUCTURES DE COMMUNICATION

Au début du paragraphe 2 et en 2.3. plus particulièrement, nous avons évoqué la généralisation du protocole d'arbitrage binaire avec ordonnancement initial PAPS au cas de n processus. Dans un système distribué, ordonnancement PAPS doit être pris, bien évidemment, avec un sens particulier lié à la perception que chaque site peut avoir des autres sites quand les délais de transmission sont a-priori inconnus, variables mais bornés (voir [COR 81]). Nous évoquons, en quelques lignes, et de manière très informelle, comment on peut généraliser de tels protocoles avec ordonnancement initial PAPS (au sens distribué).

Un premier type de solution ([LAM 74], [LAM 78]) est basé sur un maillage total : un processus communique avec ses (n-1) pairs (exemple : un bus, un réseau à diffusion). Il y a n(n-1)/2 connexions logiques. Chaque processus entre alors en compétition directe avec tous les autres. L'ordonnancement PAPS est obtenu à l'issue d'une course en ligne unique à laquelle participent tous les processus.

Un second type de solution ([KES 82]) est basé sur une structure d'arbre binaire : un processus (parmi $n = 2^p$) communique avec p processus dans une structure arborescente binaire. Il y a $\frac{1}{2} n\log_2 n$ connexions logiques (np/2). Chaque processus entre alors en compétition avec tous les autres, mais indirectement. L'ordonnancement PAPS est obtenu à l'issue d'une épreuve par élimination directe dans laquelle chaque match oppose deux processus, selon l'algorithme décrit en (IV).

Ainsi certains algorithmes distribués imposent une typologie de communication plus ou moins maillée. Cet exemple met donc bien en valeur qu'on peut "distribuer" un algorithme sur des topologies différentes et qu'il reste à faire une étude exhaustive, quant à la complexité, de différentes réalisations distribuées d'un même algorithme, comme on l'a fait des différents algorithmes de tri par exemple.

BIBLIOGRAPHIE

[AHV 80] F. André, D. Herman, J.P. Verjus
Contrôle du parallélisme et de la répartition.
Cours de l'Ecole AFCET, Aix-en-Provence - IRISA Ed. (1980).

et également :

Synchronisation de programmes parallèles, expression et mise en oeuvre dans les systèmes centralisés ou distribués.
AFCET-Informatique, DUNOD, (1983).

Synchronization of Parallel Programs.
North Holland Academic, (Feb. 1985).

[BOCH 79] G.V. Bochman
Architecture of Distributed Computer Systems.
Lecture notes in Computer Science, n° 77, Springer Verlag (1979).

et également :

Concepts for Distributed Systems Design.
Springer Verlag (1983).

[CAR 83] O.S.F. Carvalho, G. Roucairol
Assertion, Decomposition and Partial Correctness of Distributed Control Algorithm.
In Distributed Computing Systems, Y. Paker & J.P. Verjus Eds, Academic Press (1983).

[COR 81] CORNAFION
Systèmes informatique répartis, concepts et techniques.
Dunod Informatique (1981).

et également :

Distributed Computer Systems ; Communication, Cooperation, Consistency.
North-Holland (1985).

[DIJ 74] E.W. Dijkstra
Self-Stabilizing Systems in spite of distributed control.
Comm. ACM 17,11 (Nov. 1974).

[HER 83] D. Herman
Towards a systematic approach to implement distributed control of synchronization.
In Distributed Computing Systems, Y. Paker & J.P. Verjus Eds, Academic Press (1983).

[KES 82] J.L.W. Kessels
Arbitration without common modifiable variables.
ACTA INFORMATICA 17, pp 135-141, Springer-Verlag 1982.

[LAM 74] L. Lamport
A new solution of Dijkstra's concurrent programming problem.
CACM 17,8 (Aug. 74).

[LAM 78] L. Lamport
Time, clocks and the ordering of events in a distributed system.
CACM, 21, 7 (July 1978)

[MTV 77] J. Mossière, M. Tchuente, J.P. Verjus
Sur l'exclusion mutuelle dans les réseaux informatiques.
Pub. IRISA n° 75 (1977).

[PET 81] G.L. Peterson
Myths about the mutual exclusion problem.
I.P.L., 12,3, pp 115-116, (June 81).

[RAY 85] M. Raynal
Un algorithme d'exclusion mutuelle pour une structure logique en anneau.
TSI, 1985 (à paraître).

[REK 79] D.P. Reed, R.K. Kanodia
Synchronization with event counts and sequencers.
CACM 22,2 (Feb. 1979).

[ROV 77] P. Robert, J.P. Verjus
Towards autonomous description of synchronization modules.
Proc. IFIP Congress, North Holland, pp 981-986 (1977).

[VER 83] J.P. Verjus
Synchronization in Distributed Systems : an informal introduction.
In Distributed Computing Systems, Y. Paker & J.P. Verjus Eds, Academic Press (1983).

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique